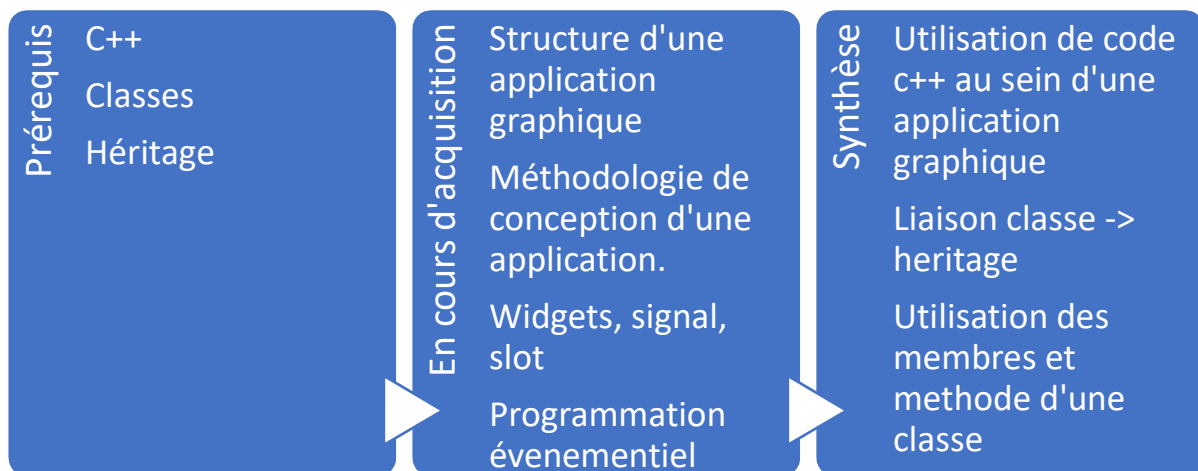


TD Conception d'un convertisseur de température avec QT



Objectifs:

- Réaliser une application graphique fonctionnelle avec QT. Cette application permettra de convertir une température de degré Celsius à Farenheit et inversement.

Evaluation:

- Autonomie
- Pertinence des recherches effectuées
- validation de chacune des étapes de la conception.
- Test et validation du produit finale



TD Conception d'un convertisseur de température avec QT

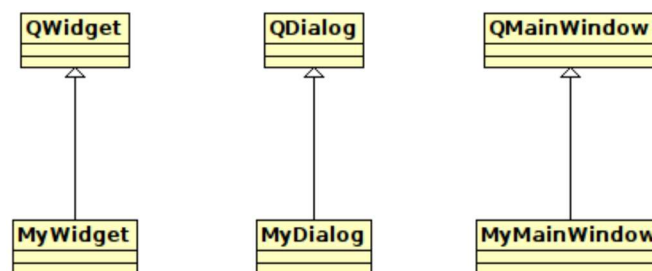
Fonctionnement

Le fonctionnement de l'application est assez simple :

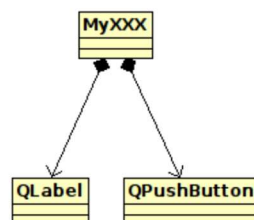
- Le bouton **Convertir** assure une conversion de la valeur saisie dans l'unité choisie
- Le bouton **Quitter** permet de terminer l'application
- Si la valeur saisie est vide, on affichera simplement “-.-” comme résultat
- Si l'utilisateur change d'unité, cela reviendra à permuter le résultat avec la valeur saisie

Principe d'une GUI QT

La création d'une fenêtre personnalisée est faite en héritant de QWidget ou QDialog ou QMainWindow :

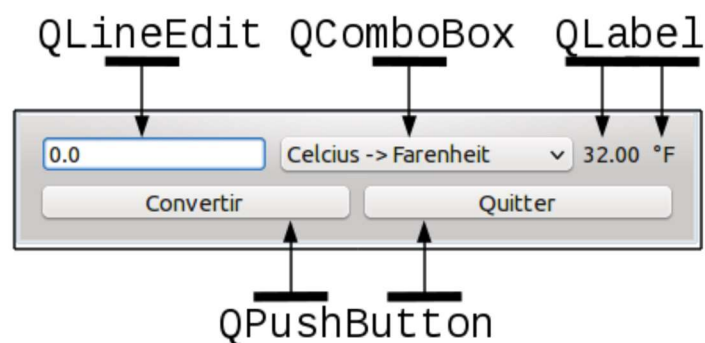


Ensuite, on compose sa fenêtre personnalisée en y intégrant des widgets :



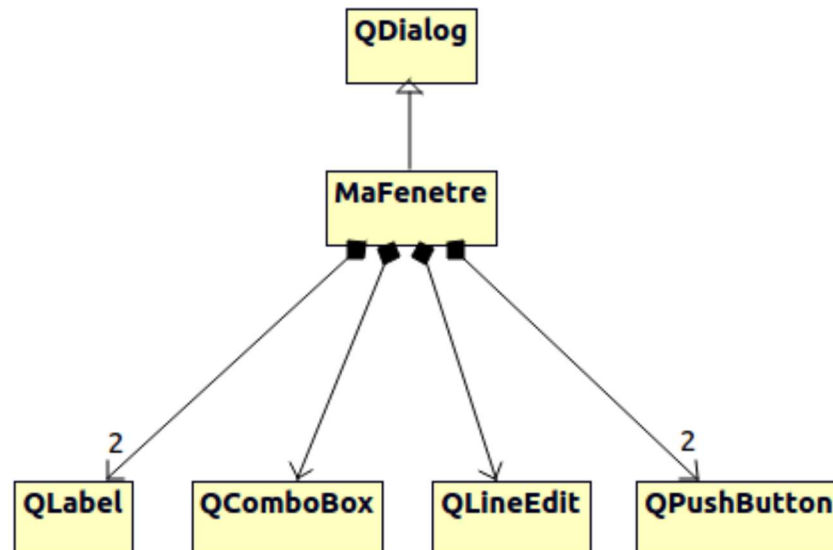
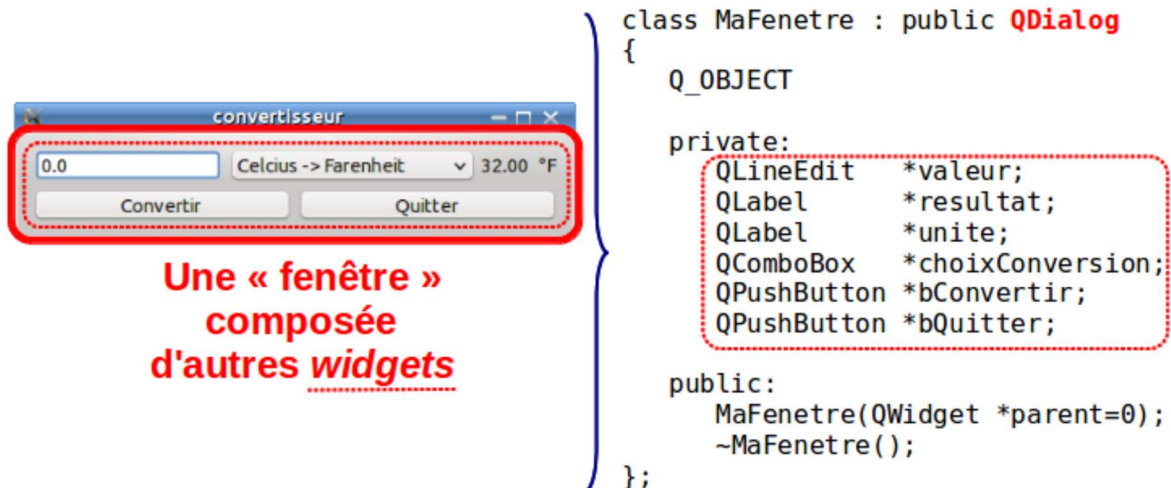
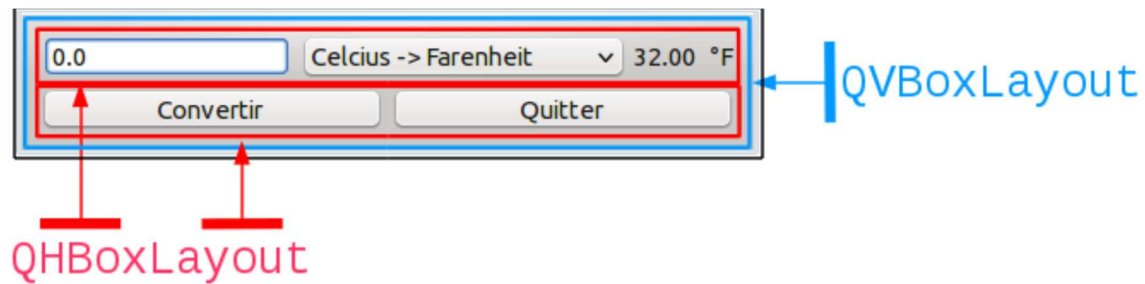
Remarque : les widgets seront positionnés dans des layouts.

La classe MaFenetre devra intégrer les différents éléments graphiques suivants :



TD Conception d'un convertisseur de température avec QT

Le positionnement de ces widgets respectera le plan suivant :



L'application sera composée des fichiers suivants :

- convertisseur.pro : le fichier de projet Qt
- main.cpp : la fonction principale main()
- mafenetre.h : la déclaration de la classe MaFenetre
- mafenetre.cpp : la définition de la classe MaFenetre

TD Conception d'un convertisseur de température avec QT

Fichier main.cpp

```
#include <QApplication>
#include "MaFenetre.h"
int main(int argc, char **argv)
{
    QApplication app(argc, argv); // un objet QApplication
    MaFenetre maFenetre; // un objet fenêtre
    maFenetre.show(); // affiche la fenêtre
    int ret = app.exec(); // exécute la boucle principale d'évènement
    return ret;
}
```

Le fichier MaFenetre.h

Le squelette de la classe MaFenetre est le suivant :

```
#if QT_VERSION >= 0x050000
#include <QtWidgets> /* tous les widgets de Qt5 */
#else
#include <QtGui> /* tous les widgets de Qt4 */
#endif
class MaFenetre : public QDialog
{
    Q_OBJECT
public:
    MaFenetre( QWidget *parent=0 );
private:
    // les widgets
signals:
public slots:
};
```


TD Conception d'un convertisseur de température avec QT

Le fichier MaFenetre.cpp

Le squelette du constructeur de la classe MaFenetre est le suivant :

```
#include "MaFenetre.h"
MaFenetre::MaFenetre( QWidget *parent ) : QDialog( parent )
{
    // 1. Instancier les widgets
    // 2. Personnaliser les widgets
    // 3. Instancier les layouts
    // 4. Positionner les widgets dans les layouts
    // 5. Connecter les signaux et slots
    // 6. Personnaliser la fenêtre
}
// 7. Définir les slots
// 8. Définir les méthodes
```

On obtient (pour l'instant) :



```
class MaFenetre : public QDialog
{
    Q_OBJECT

private:

public:
    MaFenetre(QWidget *parent=0);
    ~MaFenetre();
};

MaFenetre::MaFenetre(QWidget *parent)
: QDialog( parent )
{}

int main( int argc, char **argv )
{
    QApplication a( argc, argv );

    MaFenetre w; // rappel : pas de parent = fenêtre !
    w.show();

    return a.exec();
}
```

Tester le fonctionnement de l'application. Une fenêtre vide devrait apparaître ne vous inquiétez nous allons commencer les différentes étapes qui vont nous servir à conceptualiser l'application !!!

TD Conception d'un convertisseur de température avec QT

Etape 1 Instancier les Widgets

Remarque : il est important d'organiser ses objets Qt sous forme d'arbre car, quand Qt détruira l'objet parent, il détruira aussi tous ses objets enfants. Inutile donc de faire les delete.

a- Modification de la classe MaFenetre.h

```
#define CELCIUS_FARENHEIT 0
#define FARENHEIT_CELCIUS 1
class MaFenetre : public QDialog
{
    Q_OBJECT
    // Membre(s) public(s)
public:
    MaFenetre( QWidget *parent = 0 );

    // Membre(s) privé(s)
private:
    // Les widgets
    QLineEdit *valeur;
    QLabel *resultat;
    QLabel *unite;
    QComboBox *choixConversion;
    QPushButton *bConvertir;
    QPushButton *bQuitter;
    QDoubleValidator *doubleValidator;

};
```

b- Modification du fichier MaFenetre.cpp

```
MaFenetre::MaFenetre( QWidget *parent ) : QDialog( parent )
{
    // 1. Instancier les widgets
    valeur = new QLineEdit(this);
    resultat = new QLabel(this);
    unite = new QLabel(this);
    choixConversion = new QComboBox(this);
    bConvertir = new QPushButton(QString::fromUtf8("Convertir"), this);
    bQuitter = new QPushButton(QString::fromUtf8("Quitter"), this);
    // ...
}
```

TD Conception d'un convertisseur de température avec QT

Etape 2 Personnaliser les widgets

Toujours dans le fichier MaFenetre.cpp modifiez la partie correspondante :

```
// 2. Personnaliser les widgets
valeur->setStyleSheet("color: #0a214c; background-color: #C19A6B;");
valeur->clear();
QFont font("Liberation Sans", 12, QFont::Bold);
choixConversion->setFont(font);
choixConversion->addItem(QString::fromUtf8("Celcius -> Farenheit"));
choixConversion->addItem(QString::fromUtf8("Farenheit -> Celcius"));
resultat->setStyleSheet("color: #0a214c;");
unite->setStyleSheet("color: #0a214c;");
```

La encore tester votre application pour en voir le résultat. Soyez attentifs au changement et l'implication de chaque nouvelle instruction ajoutée au programme.

Etape 3 Instancier les layouts

```
// 3. Instancier les layouts
QHBoxLayout *hLayout1 = new QHBoxLayout;
QHBoxLayout *hLayout2 = new QHBoxLayout;
QVBoxLayout *mainLayout = new QVBoxLayout;
```

Etape 4 Positionner les widgets dans les layouts

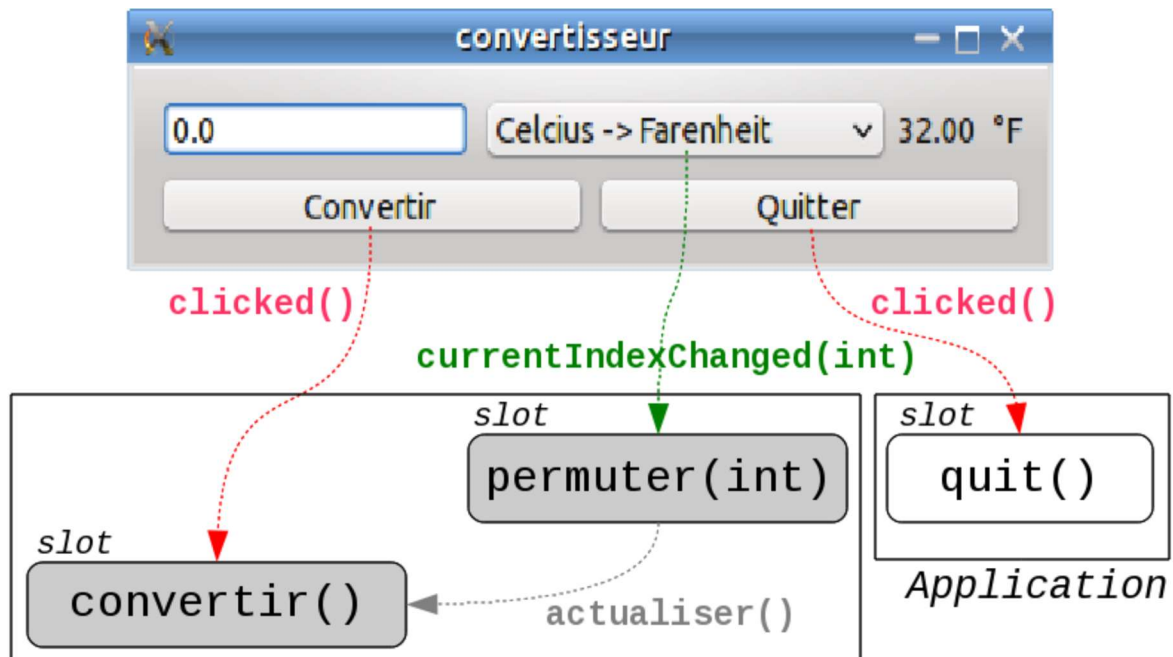
```
// ...
// 4. Positionner les widgets dans les layouts
hLayout1->addWidget(valeur);
hLayout1->addWidget(choixConversion);
hLayout1->addWidget(resultat);
hLayout1->addWidget(unite);
hLayout2->addWidget(bConvertir);
hLayout2->addWidget(bQuitter);
mainLayout->addLayout(hLayout1);
mainLayout->addLayout(hLayout2);
setLayout(mainLayout);
```

TD Conception d'un convertisseur de température avec QT

Vous devriez déjà avoir une application qui ressemble visuellement au rendu finale de notre travail cependant que constatez-vous ?

Etape 5 Connecter les signaux et les slots

Dans notre application, il faudra assurer au minimum la gestion des événements (**signaux**) et le déclenchement des actions (**slots**) représentées dans la figure ci-dessous :



Modifier le fichier MaFenetre.h afin de rajouter le code suivant :

```
#define CELCIUS_FARENHEIT 0 ←--- à rajouter
#define FARENHEIT_CELCIUS 1 ←--- à rajouter
class MaFenetre : public QDialog
{
    // .....
private:
    // .....
    void afficherUnite();
    // Mécanisme(s) Qt
signals:
    void actualiser();
private slots:
    void convertir();
    void permuter();
};
```


TD Conception d'un convertisseur de température avec QT

Dans le fichier MaFenetre.cpp modifier la partie correspondante :

```
MaFenetre::MaFenetre( QWidget *parent ) : QDialog( parent )
{
    // ...
    // 5. Connecter les signaux et slots
    connect(bConvertir, SIGNAL(clicked()), this, SLOT(convertir()));
    connect(this, SIGNAL(actualiser()), this, SLOT(convertir()));
    connect(choixConversion, SIGNAL(currentIndexChanged(int)), this, SLOT(permuter()));
    connect(bQuitter, SIGNAL(clicked()), qApp, SLOT(quit()));
    // bonus : conversion automatique
    connect(valeur, SIGNAL(textChanged(const QString &)), this, SLOT(convertir()));
    // ...
}
```

Attention à ce stade n'exécuter pas votre programme car nous devons absolument assigner les signaux aux slots tel que décrit précédemment sinon vous aurez des messages d'erreurs !!!!

Etape 6 Personnaliser la fenetre

```
MaFenetre::MaFenetre( QWidget *parent ) : QDialog( parent )
{
    // ...
    // 6. Personnaliser la fenêtre
    setWindowTitle(QString::fromUtf8("Convertisseur de températures"));
    adjustSize();
    // on lance une conversion
    //convertir();
    // ou :
    emit actualiser();
    // ...
}
```

TD Conception d'un convertisseur de température avec QT

Etape 7 Définir les Slots !

```
// 7. Définir les slots
void MaFenetre::convertir()
{
    QString temperature = valeur->text();
    if (temperature.isEmpty())
    {
        resultat->setText(QString::fromUtf8("--.--"));
        afficherUnite();

        return;
    }

    switch (choixConversion->currentIndex())
    {
        case CELCIUS_FARENHEIT:
            resultat->setText(QString::fromUtf8("%1").arg(9 *
            temperature.toDouble() / 5 + 32, 0, 'f', 2));
            break;
        case FARENHEIT_CELCIUS:
            double fahrenheit = 5 * (temperature.toDouble() - 32) / 9;
            resultat->setText(QString::number(fahrenheit, 'f', 2));
            break;
    }
}

void MaFenetre::permuter()
{
    if(resultat->text() != "--.--")
    {
        valeur->setText(resultat->text());
        emit actualiser();
    }
    afficherUnite();
}
```

Une dernière étape avant de tester tout cela puisqu'il nous reste une étape !!!!

TD Conception d'un convertisseur de température avec QT

Etape 8 définir les méthodes

```
// 8. Définir les méthodes
void MaFenetre::afficherUnite()
{
    switch (choixConversion->currentIndex())
    {
        case CELCIUS_FARENHEIT:
            unite->setText(QString::fromUtf8(" °F"));
            break;
        case FARENHEIT_CELCIUS:
            unite->setText(QString::fromUtf8(" °C"));
            break;
    }
}
```

Tester votre application et valider son fonctionnement.

Ce qu'il faut retenir :

- Pour connaître la conversion choisie, on utilisera la méthode `currentIndex()` de la classe `QComboBox`.
- La valeur saisie dans un `QLineEdit` et la valeur affichée dans un `QLabel` sont de type `QString`.
- Pour assurer les calculs de conversion, il faudra donc transformer les `QString` en double en appelant la méthode `toDouble()` disponible dans la classe `QString`. Et pour réaliser l'opération inverse (double vers `QString`), on peut utiliser la méthode `number()` ou `arg()`.
- La personnalisation d'un widget peut se faire de différente manière. On peut par exemple appeler des méthodes propres aux widgets comme `setFont()`. Qt fournit aussi le support des feuilles de style `QSS` à la manière de `CSS`.
- La structure de l'application ou son squelette doit être implémenté dans le constructeur de la classe fenêtre parente.
- Il est important d'organiser ses objets Qt sous forme d'arbre car, quand Qt détruira l'objet parent, il détruira aussi tous ses objets enfants. Inutile donc de faire les `delete`.
- Il faudra assurer au minimum la gestion des événements (signaux) et le déclenchement des actions (slots) représentées.